



UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação
Departamento de Sistemas de Computação

Implementação de uma FIFO dual
clock na FPGA

João Carlos Batista

São Carlos - SP

Implementação de uma FIFO dual clock na FPGA

João Carlos Batista

Supervisor: Daniel Marteloza Consalter

A monografia referente ao estágio realizado na empresa FIT (Fine Instrument Technology) dentro do escopo da disciplina SSC0591 do Departamento de Sistemas de Computação do Instituto de Ciências Matemáticas e de Computação – ICMC-USP.

**USP – São Carlos
2014**

*"A revolução se faz através
do homem, mas o homem
ter de forjar, dia a dia, o
seu espírito revolucionário."*

(Che Guevara)

Dedicatória

Dedico esta monografia à Universidade de São Paulo, à Moradia do Alojamento, ao Curso e aos professores que se dedicaram passando conhecimento aos alunos do Curso de Bacharelado em Informática.

Agradecimentos

É difícil de agradecer a todos que me ajudaram e me apoiaram durante o curso, por isso agradeço a todos de coração com quem convivi durante todos os anos na Universidade de São Paulo. Agradeço a todos de forma paralela.

Agradeço a Deus, pela força, saúde e por nunca me abandonar e me proteger nas horas de dificuldades.

A toda minha Família, pelo carinho, dedicação e provendo ajuda financeiras e motivacionais para a conclusão do curso.

Ao ICMC, IFSC e á USP pela oportunidade de estudo e realização do estagio, e aos professores pela paciência e dedicação.

Aos profissionais da FIT e CIERMag e a todos que me influenciaram e ajudaram na realização deste estagio, agradeço aos doutores Alberto Tannús, Mateus José Martins e Daniel Marteloza Consalter e os demais amigo de estagio.

À minha namorada Marina Jeaneth Machicao Justo, que muito além de proporcionar diversos momentos felizes ao seu lado, tem me apoiado e dado forças em uma fase difícil.

Resumo

A presente monografia tem como objetivo descrever as atividades desenvolvidas pelo aluno João Carlos Batista durante o estágio desenvolvido na empresa FIT (Fine Instrument Technology), no Laboratório da CIERMag. O desafio enfrentado neste projeto, e principal objetivo, foi de estabelecer a comunicação entre o espectrômetro da CIERMag e outros receptores, pois estes equipamentos operam com diferente frequência de clock de leitura e gravação. Neste trabalho foi desenvolvida uma estrutura de armazenamento de sinais digitais na FPGA. A implementação foi feita em linguagem VHDL, e assim foi construída uma estrutura de armazenamento de sinais (FIFO), e toda uma lógica de controle e sinalizadores que indicam os estados desta. Além disso, foi desenvolvido um ambiente de simulação para analisar o comportamento e validar o resultado de cada teste. As atividades realizadas no estágio serão descritas nesta monografia em conjunto com as funcionalidades de todos os módulos, a comunicação com outros módulos assim como também os resultados dos testes para validar a descrição da lógica do hardware.

Sumário

LISTA DE ABREVIACÕES	vii
LISTA DE FIGURAS	viii
LISTA DE TABELAS	x
CAPÍTULO 1: INTRODUÇÃO	1
1.1. A empresa onde o estágio foi realizado	1
1.2. Objetivos do Trabalho	1
1.3. Organização da Monografia	1
CAPÍTULO 2: DESENVOLVIMENTO DO TRABALHO	3
2.1. Descrição do Problema	3
2.2. Métodos, Técnicas e Tecnologias Utilizadas	3
2.3. Descrição das Atividades Realizadas	8
2.4. Resultados Obtidos	15
2.5. Dificuldades e Limitações	20
CAPÍTULO 3: CONCLUSÃO	21
3.1. Contribuições	21
3.2. Relacionamento entre o Curso e o Projeto de Estágio	21
3.3. Considerações sobre o Curso de Graduação	22
3.4. Trabalhos Futuros	22

REFERÊNCIAS.....	23
APÊNDICE A – O POTENCIAL DA FPGA.....	24
APÊNDICE B – MÓDULO DE NÚMEROS RANDÔMICOS.....	25
B.1 Implementação	25
B.2 Visão estrutural em RTL.....	27
B.3 Implementação do teste bench (bateria de teste).....	29
B.3 Simulação e Validação	30
B.4 Integração dos módulos testados.....	30
B.5 Pinagem dos pinos virtuais para os pinos reais da FPGA	31
APÊNDICE C – Implementação do módulo controlador de quantidade	33

LISTA DE ABREVIACÕES

CIERMag	Centro de Imagens e Espectroscopia in vivo por Ressonância Magnética
VHDL	Very High Speed Integrated Circuit Hardware Description Language
FPGA	Field Programmable Gate Array
FIT	Fine Instrument Technology
FIFO	Algoritmo “Primeiro em entrar- primeiro em sair” (First in First out)
RTL	Register Transfer Level
RAM	Random Access Memory
ICMC	Instituto de Ciências Matemáticas e de Computação
IFSC	Instituto de Física de São Carlos
DC-FIFO	Dual-Clock FIFO

LISTA DE FIGURAS

Figura 1 – Placa FPGA Cyclone V Soc 5CSXFC6D6F31 (vista do topo)

Figura 2 – Estrutura da FPGA

Figura 3 – Ideia geral do funcionamento da FIFO, usando código Gray nos ponteiros.

Figura 4 – Ciclo de desenvolvimento do projeto

Figura 5 – Estrutura de armazenamento FIFO intermediando a comunicação

Figura 6 – Visão do projeto FIFO em RTL

Figura 7 – Módulo Contador de Código Gray para gravação leitura

Figura 8 – Módulo Sincronismo de leitura e escrita

Figura 9 – Módulo Controlador de quantidade de dados

Figura 10 - Níveis de utilização da memória FIFO

Figura 11 – Módulo Memória FIFO

Figura 12 - Visão abstrata da memória

Figura 13 – Ambiente de teste da FIFO

Figura 14– Resultado do teste em código Gray

Figura 15– Resultado dos testes na FIFO

Figura 16 – Teste do sincronismo em conjunto com o módulo de código Gray

Figura 17 – Resultado do teste no módulo sincronismo

Figura 18 – Implementação do módulo de números randômicos em VHDL e respectiva arquitetura.

Figura 19 – Algoritmo de cálculo de números randômicos para 8 bits

Figura 20 – Visão em RTL do Projeto

Figura 21 – Área em uso do projeto

Figura 22 - Chip Planner: área em uso em detalhes

Figura 23 – Visão interna do módulos Números randômicos em RTL

Figura 24 – Implementação do test bench

Figura 25 – Resultado do test bench em Números randômicos

Figura 26 – Pin Planner

Figura 27 – Gravando o projeto na FPGA

LISTA DE TABELAS

Tabela 1 – Entradas e saídas do módulo Controlador de quantidade de dados

Tabela 2 – Condições para Leitura e gravação na FIFO

Tabela 3 – Microprocessador versus FPGA

CAPÍTULO 1: INTRODUÇÃO

1.1. A empresa onde o estágio foi realizado

O aluno é estagiário da empresa FIT (Fine Instrument Technology). A FIT é uma empresa de pesquisa, desenvolvimento e inovação focada na área de ressonância magnética. Fundada na capital paulista, em 2006, como FIT, tinha foco em soluções para ressonância magnética dentro da especialidade de radiologia médica, porém a tendência em abranger a área de ressonância magnética como um todo levou à mudança para a cidade de São Carlos, interior de São Paulo, e ao novo nome - mais apropriado para suas atividades.

Já que a empresa FIT conta com a colaboração do grupo CIERMag (Centro de Imagens e Espectroscopia in vivo via Ressonância Magnética) localizada na IFSC, no Campus I da USP - São Carlos e com a EMBRAPA Instrumentação, seus projetos visam desenvolver tecnologia para aplicar ressonância magnética na área de saúde, instrumentação científica e aplicada. Por esse motivo, o aluno estagiário, contratado pela FIT, desenvolveu suas atividades no local de trabalho da CIERMag.

1.2. Objetivos do Trabalho

O objetivo é a implementação de uma estrutura de armazenamento de sinais digitais FIFO na FPGA em linguagem VHDL. Além disso, de desenvolver um ambiente de simulação e validação, tanto individual como integrada, dos módulos da FIFO. A qual intermediará os dois equipamentos, o espectrômetro da CIERMag com outros receptores, que atuem em qualquer frequência de clock de leitura e gravação, com o intuito de aumentar o ganho de transferência de sinais digitais e evitar a perda de informação.

1.3. Organização da Monografia

A monografia está organizada em três capítulos, incluindo esta introdução no primeiro capítulo, em seguida está o segundo capítulo onde relata as atividades do estagiário na empresa, descrevendo o problema, e introduzindo os métodos, técnicas, e tecnologias utilizadas, e os respectivos resultados obtidos, assim como as dificuldades e limitações encontradas no desenvolvimento desse projeto. Finalmente, no terceiro capítulo são dadas as conclusões,

contribuições, assim como outras considerações sobre o curso, o projeto de estágio, e alguns trabalhos futuros.

CAPÍTULO 2: DESENVOLVIMENTO DO TRABALHO

2.1. Descrição do Problema

Durante o período de estágio, logo no início deparamos com um problema de comunicação de dados digitais entre dois equipamentos que trabalham com diferentes frequências de clock. Assim, o problema geral foi quebrado em vários subproblemas que foram tratados como módulos que assumem um comportamento seguindo a lógica do projeto da CIERMag.

A estratégia é montar uma estrutura de armazenamento de sinais digitais implementados na FPGA. Dita estrutura deve permitir gravar em qualquer frequência de clock, ou seja, receber sinais numa frequência qualquer e armazenar estes dados numa estrutura que possa ser resgatada, como é o caso quando o receptor recebe os dados numa outra frequência. Assim, é necessária a implementação de uma estrutura de armazenamento de sinais digitais FIFO, assim como a simulação e testes da FIFO.

O estagiário concentrou a atenção na implementação da estrutura de armazenamento de sinais FIFO, que é fundamental para a comunicação entre o emissor e receptor, para o qual foram estudadas várias metodologias e, além disso, foram envolvidas diversas tecnologias que serão descritas a seguir.

2.2. Métodos, Técnicas e Tecnologias Utilizadas

2.2.1. Ambientes de desenvolvimento

Toda a implementação foi feita na plataforma de desenvolvimento Quartus II 64 bit versão 14.0 web edition da empresa ALTERA, fabricante de dispositivos lógicos programáveis. O Quartus é integrado com um editor, compilador, simulador e ferramenta para síntese. Também foi usado, o ambiente de simulação ModelSim Starter Edition 10.1 da própria ALTERA, e outros testes foram realizados no kit de desenvolvimento FPGA Cyclone V Soc 5CSXFC6D6F31.

A Figura 1 mostra todas as entradas e saídas reais da FPGA. Mais adiante será mostrada a comunicação das entradas e saídas virtuais criadas nos módulos de VHDL com as entradas e saídas reais da placa, e a associação de pinos.

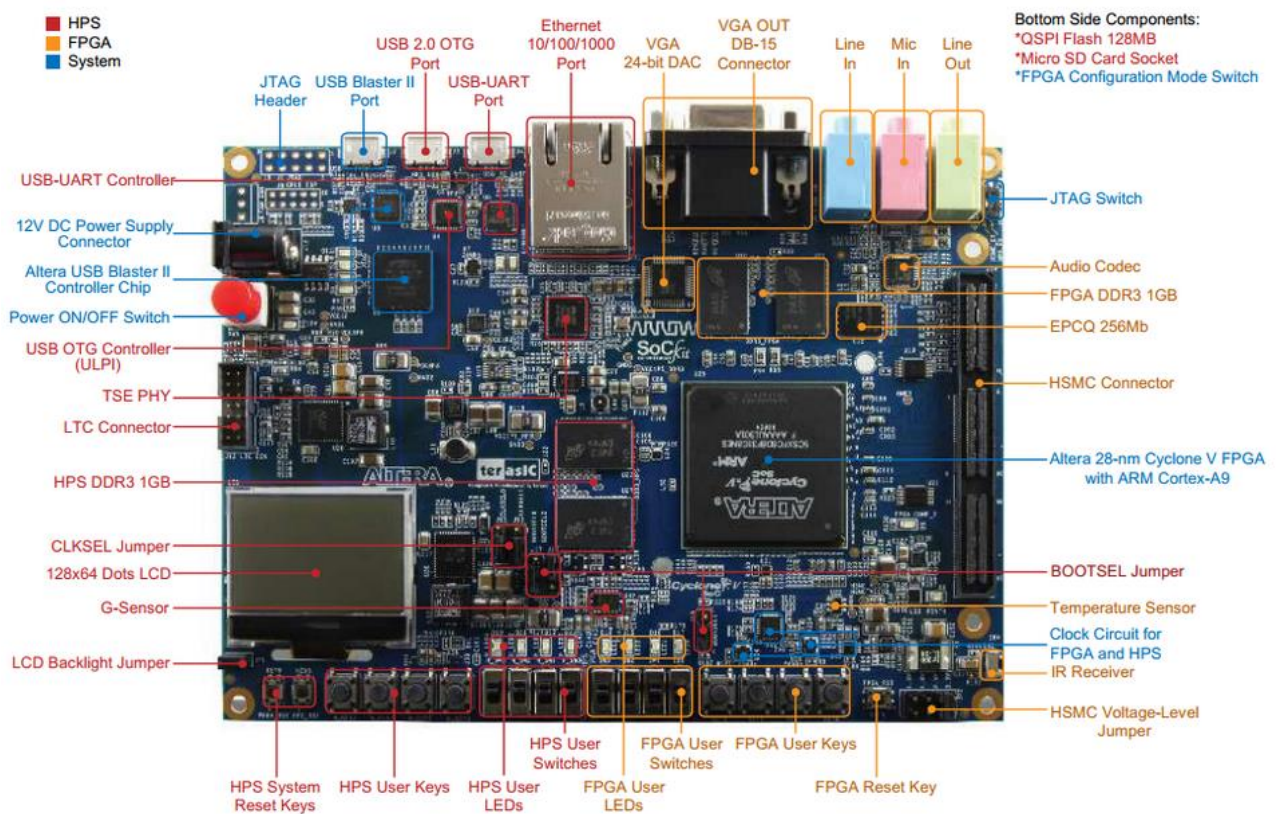


Figura 1 - Placa FPGA Cyclone V Soc 5CSXFC6D6F31 (vista do topo) [6]

O uso das FPGA é amplamente vantajoso, pois possibilita realizar inúmeras instruções por cada ciclo de clock. No Apêndice A são descritas as diversas vantagens da FPGA em comparação com os microprocessadores.

2.2.3. Estrutura da FPGA

A FPGA é um chip que implementa a função lógica gerada a partir de código em linguagem VHDL. No interior contém células lógicas programáveis, combinadas por uma matriz de vários circuitos lógicos, interligadas de diversas formas.

A estrutura da FPGA pode ser observada na Figura 2. Dentro dos blocos lógicos programáveis existem tabelas lógicas a qual está relacionada às portas logicas XOR, NAND, NOT, AND,..., flip-flop, multiplexadores, entre outros, nas quais são gravadas as funções

lógicas. Os chaveadores (switch box) de cada bloco lógico estão ligados a outros chaveadores por meio de uma matriz de array. O chaveador é reconfigurado para conectar-se a outro chaveador seguindo a lógica dos blocos lógicos programáveis, formando uma rede de interconexões.

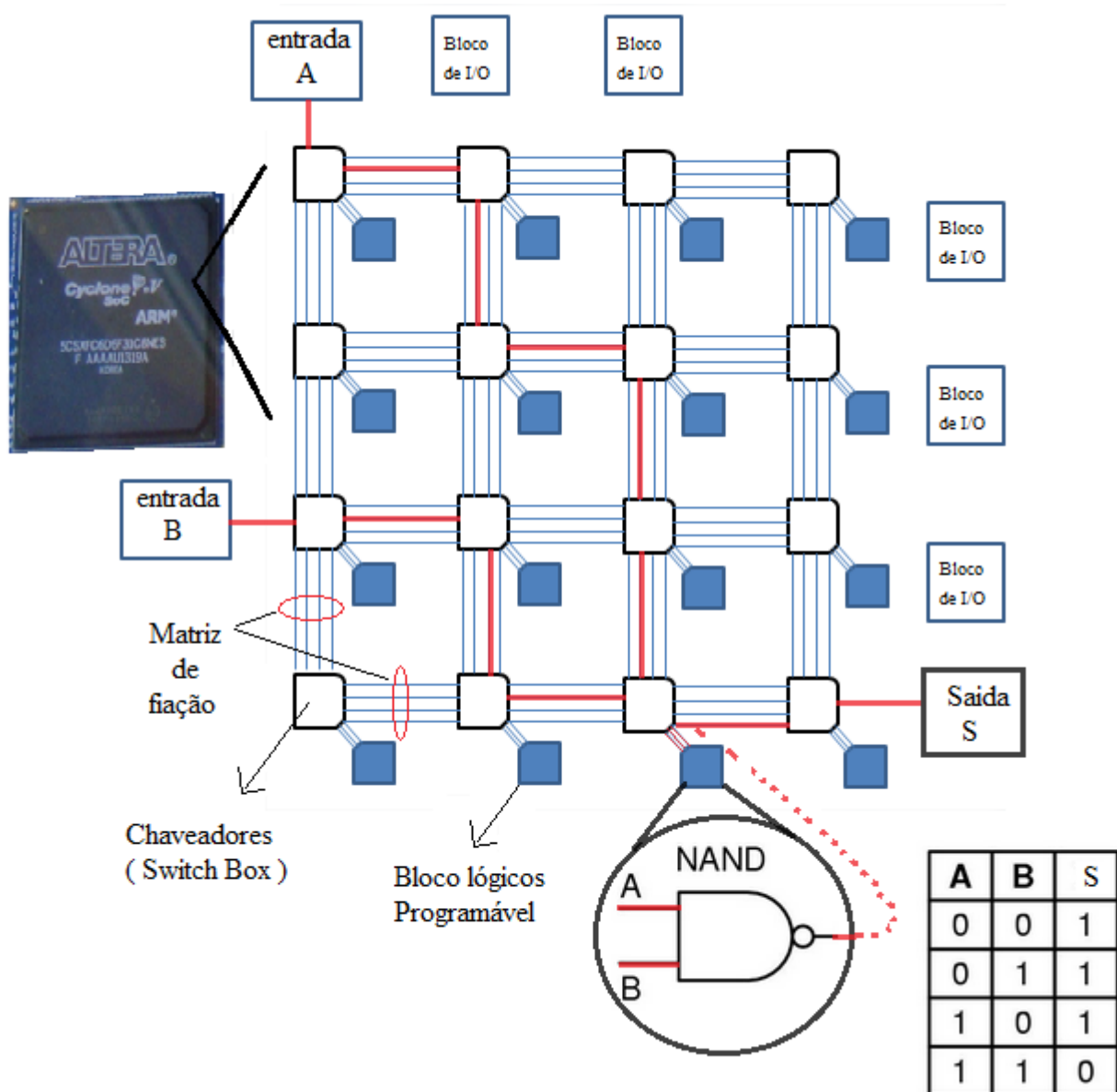


Figura 2 – Estrutura da FPGA (modificado a partir de [2,6])

2.2.4. Linguagem VHDL

O VHDL é uma linguagem padronizada para descrever os componentes digitais. Com o VHDL é permitido a modelagem do hardware e construção de circuitos mais complexos

fazendo uso das inúmeras portas lógicas que podem ser combinadas. Esta linguagem permite a programação em alto nível de abstração, a redução de tempo e custo de desenvolvimento, e também permite ser independente da tecnologia usada [4].

Observe-se que para facilitar a compreensão no decorrer da monografia, as **variáveis** em linguagem VHDL, serão marcadas com diferente estilo.

2.2.5. FIFO dual clock

Em geral, uma estrutura de dados FIFO admite a inserção e remoção de dados seguindo uma política de primeiro dado inserido é também o primeiro dado a ser removido [2]. Já a arquitetura de um FIFO dual clock implementada em Hardware segue uma política na qual existem dois clock concorrentes de endereços na memória (leitura e escrita) operando em diferentes frequências de clock [1], como é mostrado na Figura 3.

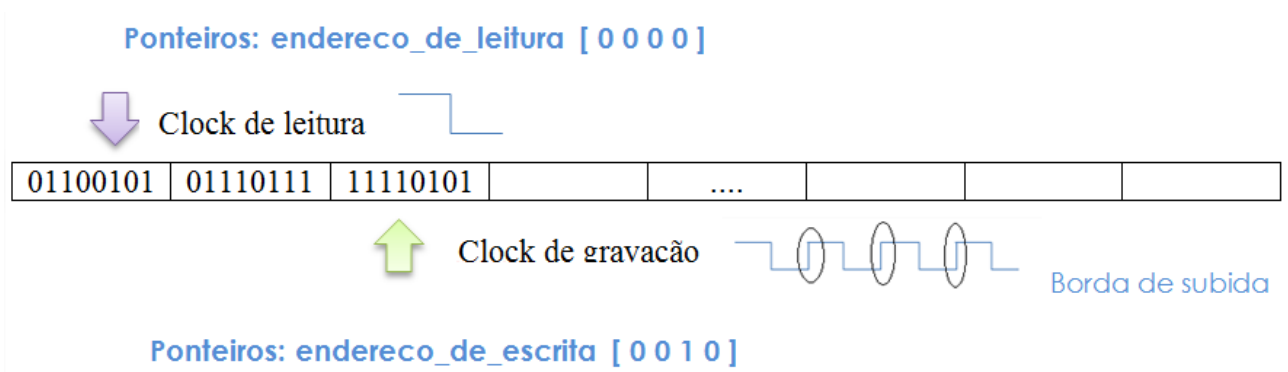


Figura 3 - Ideia geral do funcionamento da FIFO, usando código Gray nos ponteiros.

Observe-se que o ALTERA possui uma estrutura chamada DC-FIFO, a qual é paga e fechada [1], por este motivo a FIT propôs a própria implementação.

2.2.5. Ciclo de desenvolvimento de sistemas digitais em VHDL

Neste projeto é importante aprimorar o processo de desenvolvimento. Deste modo, em geral, segue-se a metodologia de ciclo de desenvolvimento, que é dividido em várias etapas anteriores à configuração da FPGA até a gravação. Esta metodologia permite ter um aumento na produtividade do desenvolvedor, na qualidade do desenvolvimento, modularização, manutenção, e facilitando a realização de testes e correção do código [5], as quais são:

1. Captação de sinais: Recebe sinais de um gerador de sinais randômicos
2. Implementação do módulo VHDL: implementando a descrição do Hardware
3. Visão estrutural em RTL: mostra a interpretação da lógica implementada no Quartus.

4. Implementação da bateria de testes (*test bench*): gera sinais de teste para as entradas dos diferentes módulos.
5. Simulação e Validação: Analisa os resultados do comportamento do módulo testados
6. Integração dos módulos testados: integrar o módulo desenvolvido ao projeto
7. Pinagem dos pinos virtuais para os pinos reais da FPGA: Fazer as conexões entre as entradas e saídas virtuais do projeto em VHDL com as pinagem reais da FPGA
8. Gravação na FPGA.

Na Figura 4 é mostrado o ciclo de desenvolvimento que seguirá este projeto. Para melhor compreensão do ciclo de desenvolvimento é exemplificado e detalhado o módulo de números randômicos no Apêndice B.

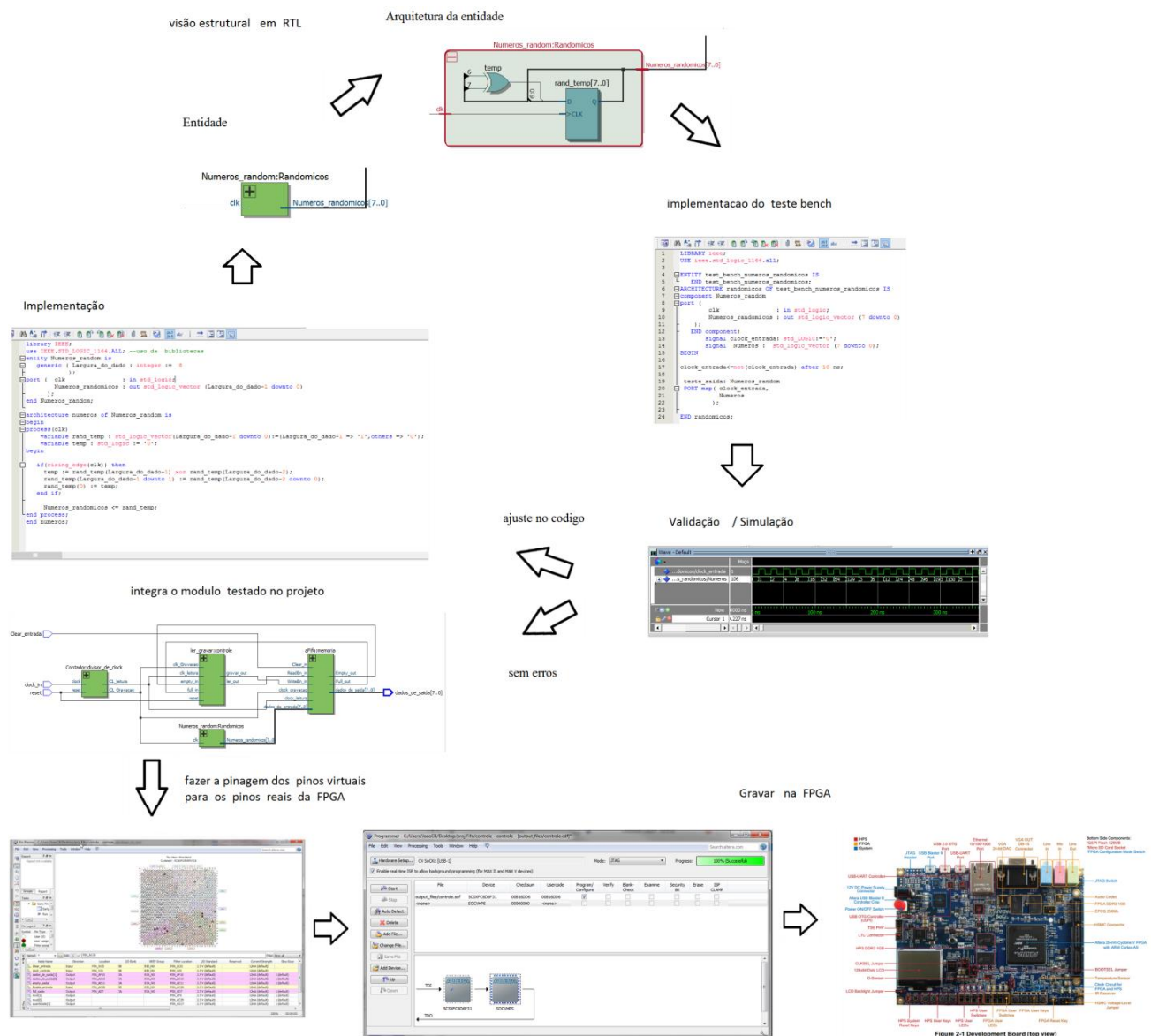


Figura 4 – Ciclo de desenvolvimento do projeto

2.3. Descrição das Atividades Realizadas

Durante o estágio realizado na FIT, foi feito inicialmente um estudo da linguagem VHDL junto como uma análise do problema de comunicação entre dois equipamentos que operam em diferentes frequências de clock.

Assim foi desenvolvida uma estrutura de armazenamento FIFO dual clock com seus respectivos mecanismos de controle e verificação de estados. Esta FIFO está composta de diversos módulos VHDL que seguem um ciclo de desenvolvimento em comum (Seção 2.2), mudando somente as portas de entradas e saídas e o comportamento respectivo de cada módulo resultante da implementação lógica da arquitetura. A seguir, será descrito a estrutura de armazenamento FIFO que intermediará a comunicação entre os dois equipamentos (espectrômetro e outros receptores).

2.3.1. Visão da FIFO dual clock

A Figura 5 mostra a visão geral do projeto. A estrutura FIFO vai intermediar a comunicação entre o emissor e o receptor. O emissor de sinais digitais (Espectrômetro) envia dados 0s e 1s numa certa frequência de clock de gravação por um canal de comunicação que pode ser fibra ótica, rede, entre outras, para logo estes dados serem armazenados na FIFO. Do outro lado, o receptor recebe estes dados com outra frequência de clock de leitura.

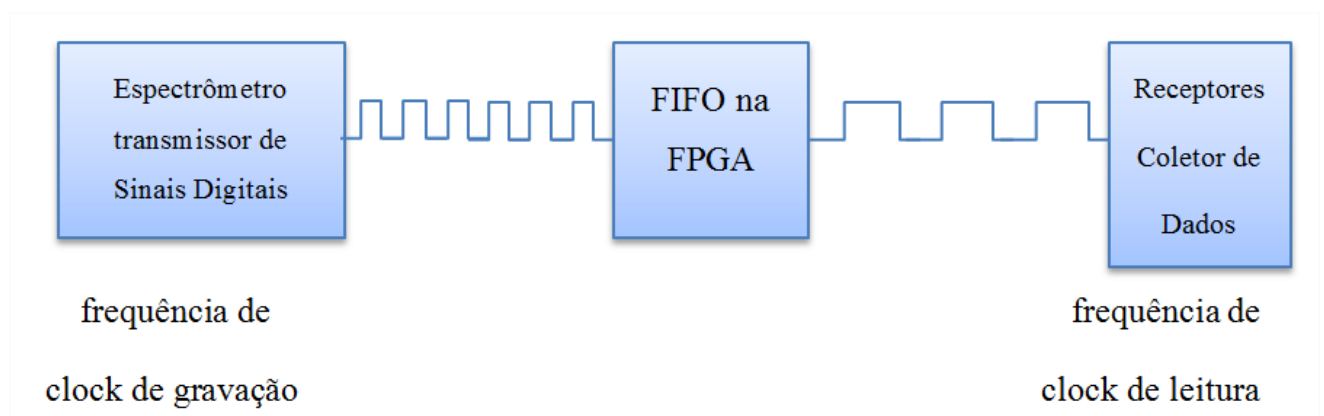


Figura 5 – Estrutura de armazenamento FIFO intermediando a comunicação

A Figura 6 mostra os quatro módulos que compõem a FIFO dual clock desenvolvida, os quais são: (i) Contador de Código Gray, (ii) Sincronismo, (iii) Memória FIFO e (iv) Controlador de quantidade de dados, os quais serão detalhados a seguir.

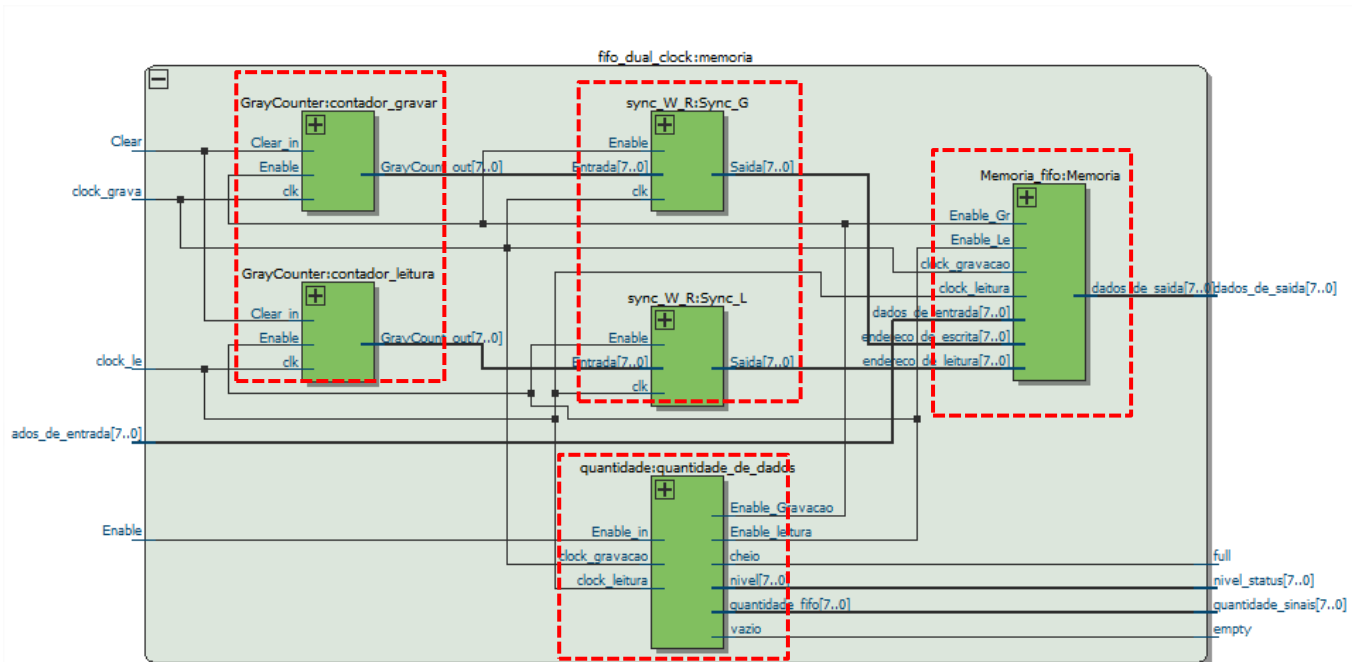
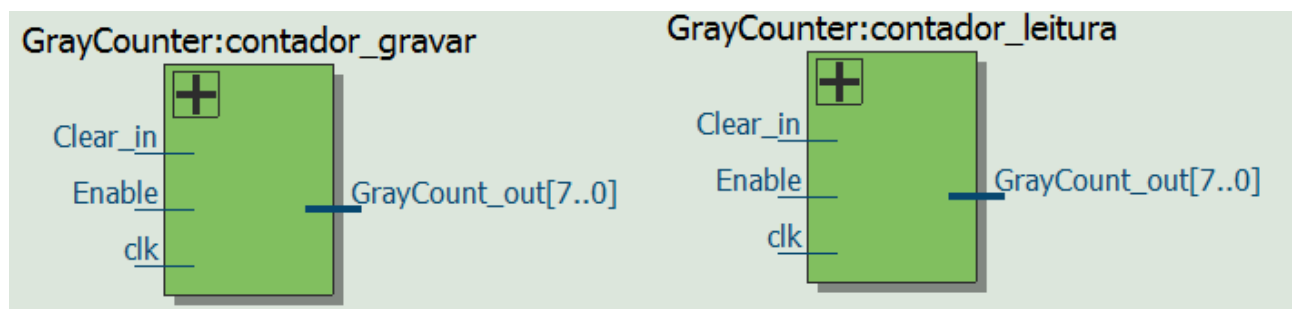


Figura 6 – Visão do projeto FIFO em RTL

2.3.2. Módulo Contador de Código Gray

O módulo **GrayCounter** é um contador em código Gray¹, observado na Figura 7. Este mesmo módulo é usado para endereçamento de leitura e gravação (**contador_leitura** e **contador_gravar**) foi instanciado para o uso no módulo Memória FIFO. Em ambos os casos, a funcionalidade da lógica implementada é a mesma só diferindo nos sinais de saída (**GrayCount_out**) que são influenciados pelos sinais de entrada (**Clear_in**, **Enable**) e principalmente pelo clock (**clk**) [7].



¹ O sistema de código binário de Gray é tal que a diferença de um número para outro número consecutivo muda um único bit.

Figura 7 – Módulo Contador de Código Gray para gravação leitura

Considerando as seguintes configurações de entrada:

- **Clear_in** = '1' o contador é inicializado, não ocorrendo contagem;
- **Enable** = '1' habilita a contagem;
- **Clear_in** = '0' é o clock oscilando entre zeros e uns,

Obtém-se um novo número gerado. O número servirá para endereçar os dados na memória (endereçamento de leitura e escrita).

Observe-se que o código Gray é empregado para evitar erros de endereçamento e transmissão de dados, em sistemas que operam em alta velocidade, no cenário em que varias condições de entrada variam ao mesmo tempo, como é o caso dos clock de leitura e gravação.

2.3.3. Módulo Sincronismo

O módulo **sync_W_R** é responsável de fazer o sincronismo de endereçamento (leitura e gravação) no módulo Memória FIFO. Observando a Figura 6, este mesmo módulo foi instanciado para **sync_W_R:Sync_L** (leitura) e **sync_W_R:Sync_G** (gravação).

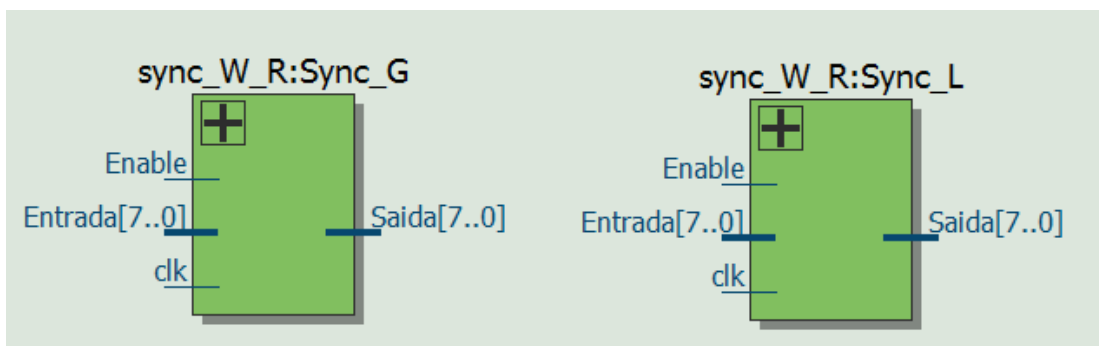


Figura 8 – Módulo Sincronismo de leitura e escrita

O sinal de **Enable**= '0' desabilita a funcionalidade do módulo, caso contrário, a saída fica sensível ao clock. Durante o primeiro clock armazena-se o dado numa variável interna, logo no segundo clock esta variável interna atribui os dados na saída final. Note-se que o primeiro dado só aparecerá na saída logo após o segundo clock, por causa do sincronismo implementado. O disparo do evento ocorre na borda de subida do clock.

2.3.4. Módulo Controlador de quantidade de dados

O módulo **quantidade_de_dados** encarrega-se de gerenciar o comportamento da FIFO em geral. Como pode ser observado na Figura 9, este módulo permite sinalizar os estados

da FIFO: **cheio**, **vazio** e **nível** de utilização, controlando o **enable_gravação** e **enable_leitura**. Este módulo também se encarrega de habilitar os outros módulos que compõem a FIFO, já que estão intercomunicados como foi mostrado na Figura 6. O código em VHDL deste módulo é mostrado No Apêndice C.

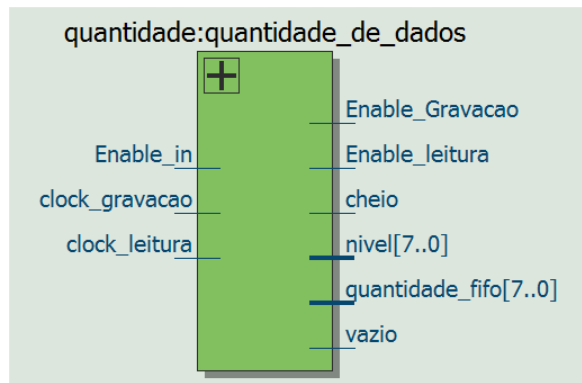


Figura 9 – Módulo Controlador de quantidade de dados

Os dados de entrada e saídas deste módulo são detalhados na Tabela 1.

Tabela 1: Entradas e saídas do módulo Controlador de quantidade de dados

	Descrição
Entradas	
Enable_in	A entrada Enable_in = '1' desabilita as funcionalidades de todos os módulos ligados dentro da FIFO, é dizer, tem o comportamento de habilitação geral, e desativa quando Enable_in = '0'. Porém, o clock continua sendo distribuído para os demais módulos.
clock_gravacao , clock_leitura	Entradas que recebem os sinais que provem do simulador de frequência dos dois equipamentos.
Saídas	

quantidade_FIFO	<p>Mostra a quantidade de sinais (número de dados) que foram armazenados na FIFO, após a cada leitura e gravação. Esta quantidade é regulada pela subtração entre os números de dados gravados e os números de dados lidos. Os sinalizadores de estado FIFO são as seguintes:</p> <ul style="list-style-type: none"> • nível: indica os níveis de utilização 1, 2, 3 e 4 (níveis de utilização da memória FIFO) • vazio= '1' : indica que a FIFO não tem elementos. • Cheio= '1': indica que a FIFO atingiu a capacidade máxima. <p>O resultado destas variáveis dependerá diretamente da variável quantidade_FIFO</p>
Enable_leitura, Enable_Gravacao	Desabilita as funcionalidades dos módulos, contador_leitura e contador_gravar respectivamente.
Sync_L, Sync_G	São sinais de leitura e gravação implementadas no módulo memória FIFO respectivamente.

O resultado assumido por essas variáveis serão tratados em seis processos dedicados, os quais são:

1. **Indicador de nível**: Recebe os sinais clock de leitura e gravação. Este processo sinaliza o estado **nível** com a quantidade de dados armazenados na FIFO, por meio de uma comparação com os intervalos em relação com a capacidade da FIFO que é dividida em quatro partes (ver Figura 10).

Nível 1	$< \frac{1}{4}$
Nível 2	$\frac{1}{4}$ e $\frac{1}{2}$
Nível 3	$\frac{1}{2}$ e $\frac{3}{4}$
Nível 4	$> \frac{3}{4}$

Figura 10 - Níveis de utilização da memória FIFO

2. **Avaliar o estado da FIFO**: O estado de utilização da FIFO deve ser verificado a cada instante, pois é o momento crítico da FIFO. A cada mudança ocorrerá o disparo deste processo, onde são verificadas as seguintes políticas:

- a) Estado FIFO vazia: Caso seja menor que 1 a variável **vazio** = '1' indicando que a FIFO está vazia e '0' caso contrário.
- b) Estado FIFO cheia: caso a profundidade da FIFO seja igual á capacidade máxima o valor **cheio** = '1' indicando FIFO cheia, e '0' caso contrário.

Note-se que os estados da FIFO, **cheio** ou **vazio**, são inicializados com zero, para garantir todas as possibilidades de estado.

3. **Incrementa contador após gravação:** Neste processo recebe-se o clock de gravação na lista de sensibilidade, a cada borda de subida de clock. Encarrega-se de incrementar o contador da FIFO se acaso a seguinte condição for válida **Enable_G** = '1' e **grava** = '1'.

4. **Incrementa contador após leitura:** Este processo segue a mesma lógica que o anterior, usando o clock de leitura como estímulo e as condições necessárias para incrementar o contador da FIFO se a condição **Enable_L** = '1' e **le** = '1' é válida.

Observe-se que o resultado após incrementar ambos os contadores de gravação e leitura, facilitam os cálculos de quantidade de dados na FIFO (ver o APÊNDICE C), já que se segue o critério **quantidade_fifo** = **incrementar_gravacao** - **incrementar_leitura**.

5. **Controlar leitura e gravação:** Este processo recebe o **clock_gravacao** e **clock_leitura** na lista de sensibilidade. Este processo é responsável de controlar as condições em que a FIFO deve gravar ou ler um sinal digital. Em geral, só é possível ler a FIFO caso haja pelo menos um elemento nela e também só é possível gravar quando a FIFO não esteja cheia.

a) A variável **gravar** assume valor '1' quando a FIFO não atingiu a sua capacidade máxima e '0' caso contrário. Este sinal é responsável de incrementar o contador de gravação, habilitando ou desabilitando a gravação respectivamente.

b) A variável **le** assume valor '1' quando a FIFO tem mais de um elemento e '0' caso contrário. Este sinal é responsável de incrementar o contador de leitura, habilitando ou desabilitando a leitura respectivamente.

6. **Semáforo de leitura e gravação:** Este processo recebe o **clock_gravacao** e **clock_leitura** na lista de sensibilidade. Este processo é responsável de habilitar ou desabilitar a gravação ou leitura da FIFO. Este mesmo processo também encarregasse de enviar sinais de controle para os módulos externos Contador de Código Gray, Sincronismo e Memória FIFO. As notificações de sinais de controle são **Enable_Gravacao** e **Enable_Leitura** para os

módulos externos, além de controlar os processos de incrementa gravação e incrementa leitura interno deste módulo usando os **Enable_G** e **Enable_L**.

2.3.5. Módulo Memória FIFO

O módulo **Memoria_FIFO** é responsável por armazenar os sinais, tem com entrada os sinais de controle que permite a leitura ou a gravação de dados como pode ser observado na Figura 11.

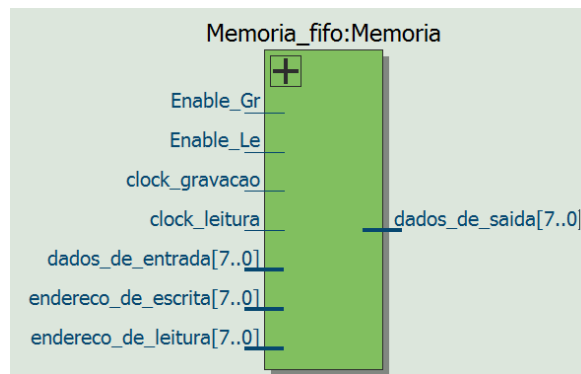


Figura 11 – Módulo de Memória FIFO

Para que ocorra a gravação ou leitura de um sinal digital devem ser cumpridas certas configurações dadas na Tabela 2.

Tabela 2 – Condições para Leitura e gravação na FIFO

Para que ocorra a gravação de um sinal digital	Para que ocorra a leitura de um sinal digital
Condições: A FIFO não deve estar cheia. O Enable_Gr ='1'. O sinal digital preparado para ser gravado. A mudança do clock_gravacao de '0' para '1'.	Condições: A FIFO não deve estar vazia. O Enable_Le ='1'. A mudança do clock_leitura de '0' para '1'.

Para fim de melhor compreensão, a memória FIFO é representada por uma matriz. Observando a Figura 12, mostra-se o código Gray endereçado, assim como a largura do sinal digital armazenado e a capacidade máxima de armazenamento.

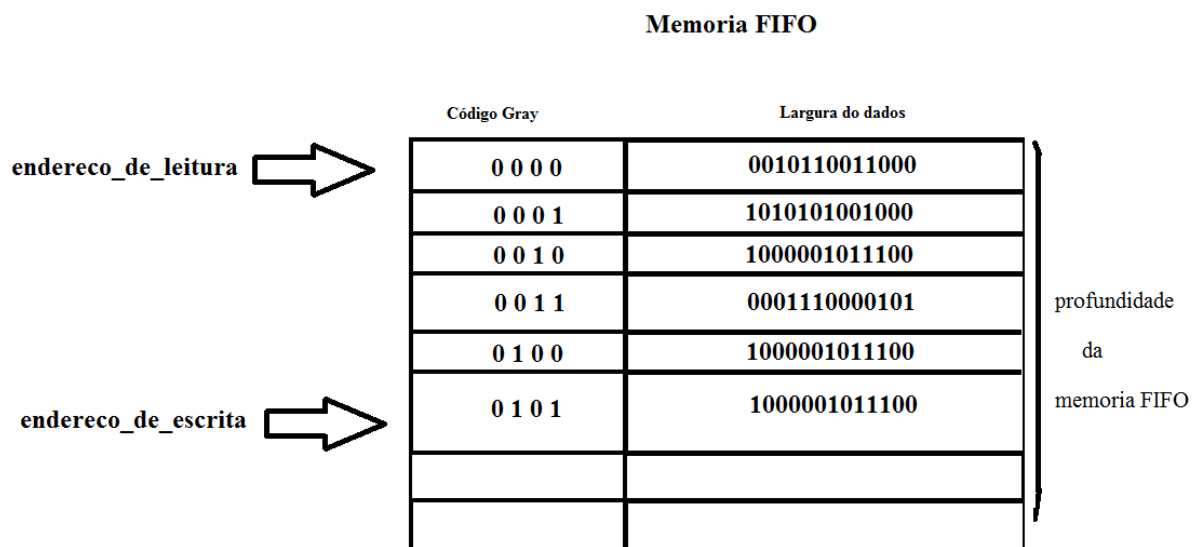


Figura 12 - Representação da memoria FIFO

2.4. Resultados Obtidos

De forma a se validar a FIFO implementada, foi necessário a simulação e execução de uma bateria de testes apresentando cenários diferentes. Esperasse que os quatro módulos da FIFO, tanto individuais como integradamente, atinjam resultados positivos, para logo após serem testados na placa ALTERA.

Uma bateria de testes em VHDL foi criada por cada módulo desenvolvido. A implementação destes testes tem como objetivo de (a) introduzir sinais para cada entrada dos módulos, (b) forçar ao erro, (c) buscar pontos de falhas, para assim analisar os sinais de saída esperados ou no caso lançar exceções.

Para a simulação destes cenários foram implementados dois módulos a seguir: (i) o módulo de números randômicos, que fornece a entrada de dados da FIFO (detalhada amplamente no Apêndice B), assim como de (ii) o módulo divisor de clock que representará a frequência de clock dos dois equipamentos (espectrômetro e outros receptores), gerando sinais com diferente clock de leitura e gravação e diferente largura de dado.

Na Figura 13, observa-se por um lado ao emissor como um gerador de sinais digitais (números randômicos) que a sua vez recebe entradas do divisor de clock, e do outro lado o receptor a FIFO dual clock.

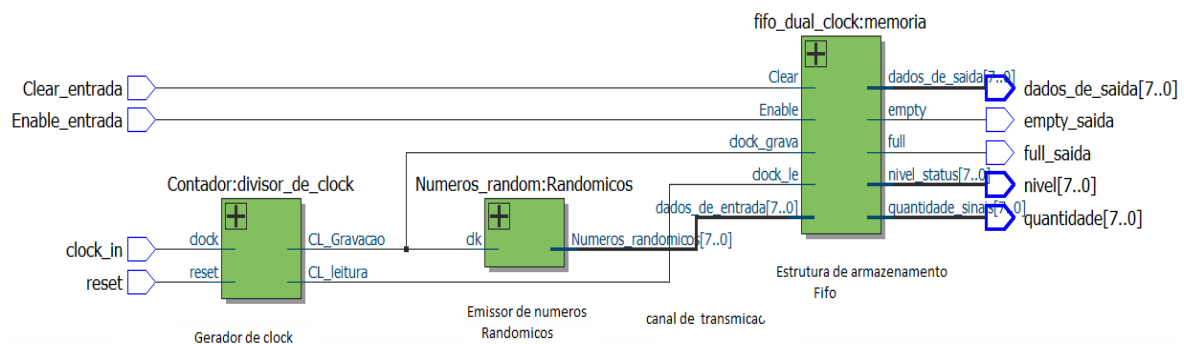


Figura 13 – Ambiente de simulação de teste da FIFO

A seguir serão apresentados vários cenários de testes com situações que induzem a erro e o correspondente comportamento esperado da FIFO.

2.2.4. Cenário 1: Módulo do contador Código Gray

A Figura 14 mostra o resultado da simulação do módulo de Contador de Código Gray, onde é observada a mudança de um único bit para cada número consecutivo.

- Para inicializar o contador deve-se ter pelo menos um clock com as variáveis configuradas como **enable=0** e **clear=1**.
- Para iniciar a contagem devem-se ter os clock e as variáveis configurados como **enable=1** e **clear=0**.

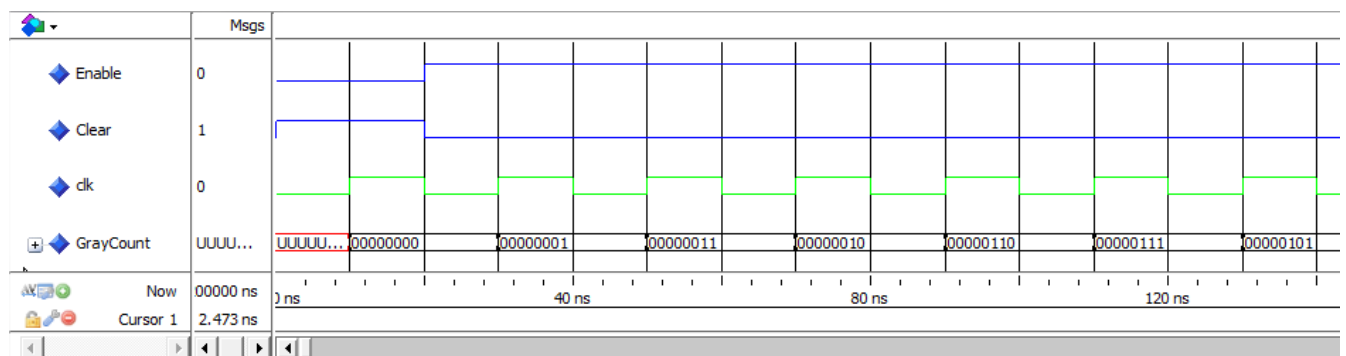


Figura 14– Resultado do teste em código Gray

Observe-se que quando o contador de código Gray atinge seu valor máximo, este é zerado e continua contando.

2.2.5. Cenário 2: Teste na FIFO integrada

A Figura 15 mostra o resultado do teste na FIFO já integrada. Para melhor compreensão e visualização dos resultados, a FIFO foi configurada com uma capacidade de armazenamento

de 16 sinais e uma largura de dados de 4 bits. Nota-se que o clock de leitura e gravação está mudando ao longo desse teste, a qual representa situações adversas dos equipamentos (emissores e receptores): só gravar e não ler dados, só ler dados e não gravar ou gravar e ler dados juntos, porém com frequência de leitura e gravação diferentes.

O início das operações de leitura e escrita de dados somente é habilitado quando ocorrem as seguintes configurações:

- As condições para inicializar o contador de Código Gray são **Clear**='1' e **Enable**= '0', em seguida pelo menos um clock de leitura e um clock de gravação. Isto é mostrado na Figura 15 região 1.
- As condições para que haja a leitura ou escrita dos dados são **Clear**=0 e **Enable**=1, em seguida os demais clock.
- Os sinalizadores que indicam status da FIFO são: **cheio**, **vazio**, **nível** e **Quantidade**.

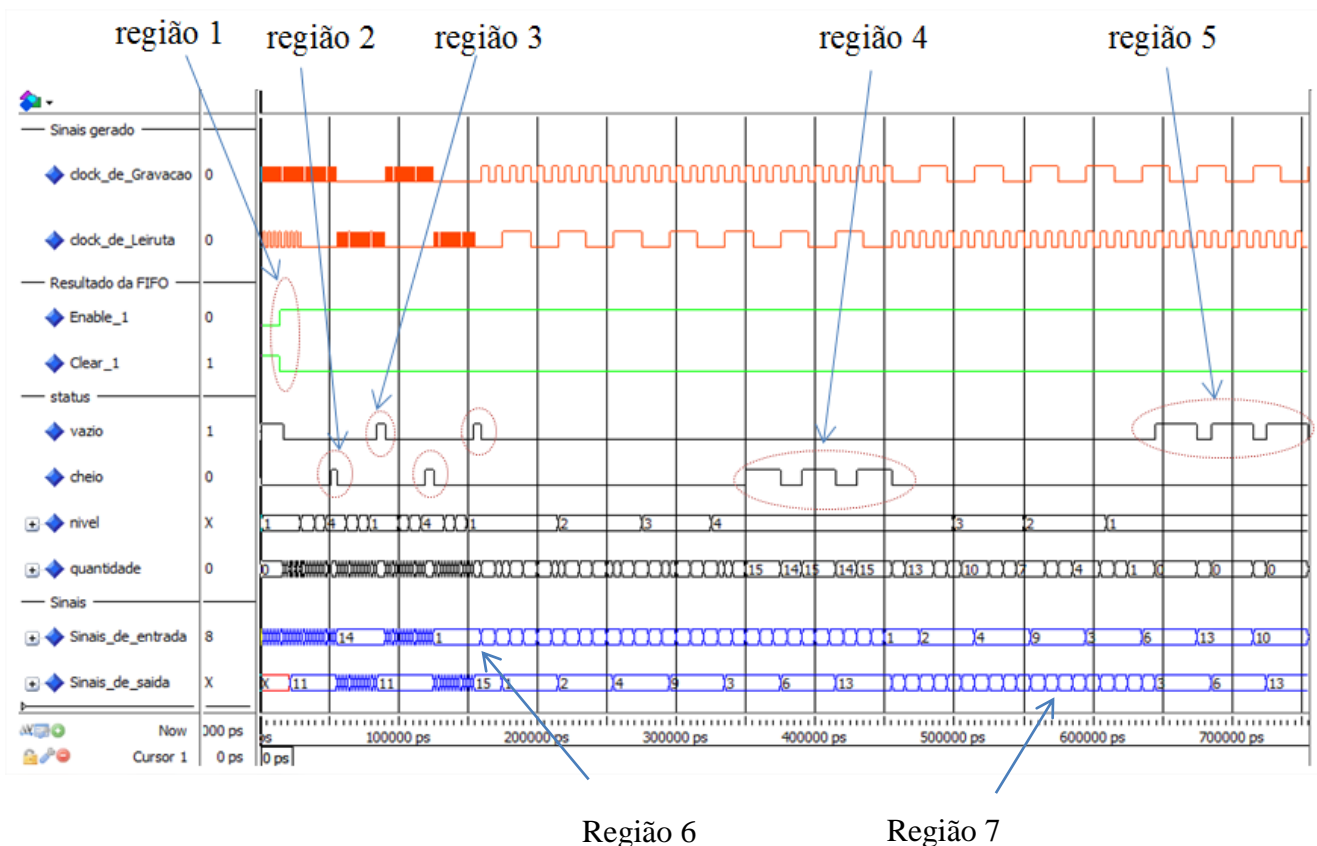


Figura 15– Resultado dos testes da FIFO integrada usando o ModelSim

Analisando as regiões da Figura 15 (círculos), observam-se os seguintes estados:

- Na região 2 o estado de FIFO cheia.
- Na região 3 o estado de FIFO vazia.
- Na região 4 o estado de FIFO cheia, devido a que o clock de gravação tem uma frequência maior que a frequência do clock de leitura, é possível notar que os valores de quantidade varia de 15 a 14.
- Na região 5 o estado de FIFO vazia, devido a que o clock de gravação tem uma frequência menor que a frequência do clock de leitura, por esse motivo o valor de quantidade varia de 0 a 1.
- Na região 6 a FIFO está recebendo sinais do espectrômetro ou de outros transmissores de sinais digitais.
- Na região 7 a FIFO está enviando os sinais armazenados para os receptores.

O resultado mostra que o transmissor pode enviar dados em qualquer frequência de clock de gravação, o qual é só permitido se a FIFO não estiver cheia, por outro lado, o receptor pode ler estes dados também em qualquer frequência de clock de leitura, o qual é só permitido se a FIFO não estiver vazia.

Observe-se que a FIFO pode ter largura de dados variando de 4 até 32 bits (valores múltiplos de 2) e a capacidade de armazenamento também ajustáveis, variando de 2^2 a 2^{20} . Vale lembrar que podemos instanciar vários módulos FIFO e rodar em paralelo.

2.2.6. Cenário 3: Teste de Sincronismo

Para a realização deste teste foi necessária a junção de dois módulos, do módulo Sincronismo que recebe a saída do **GrayCount_out** e do módulo Contador de Código Gray, ambos estão no mesmo clock de gravação, como se observa na Figura 16.

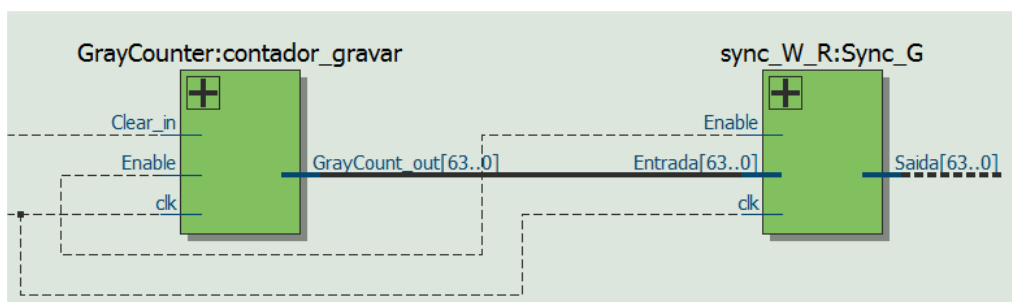


Figura 16 – Teste do sincronismo em conjunto com o módulo de código Gray

O resultado da simulação do módulo sincronismo encontra-se na Figura 17.

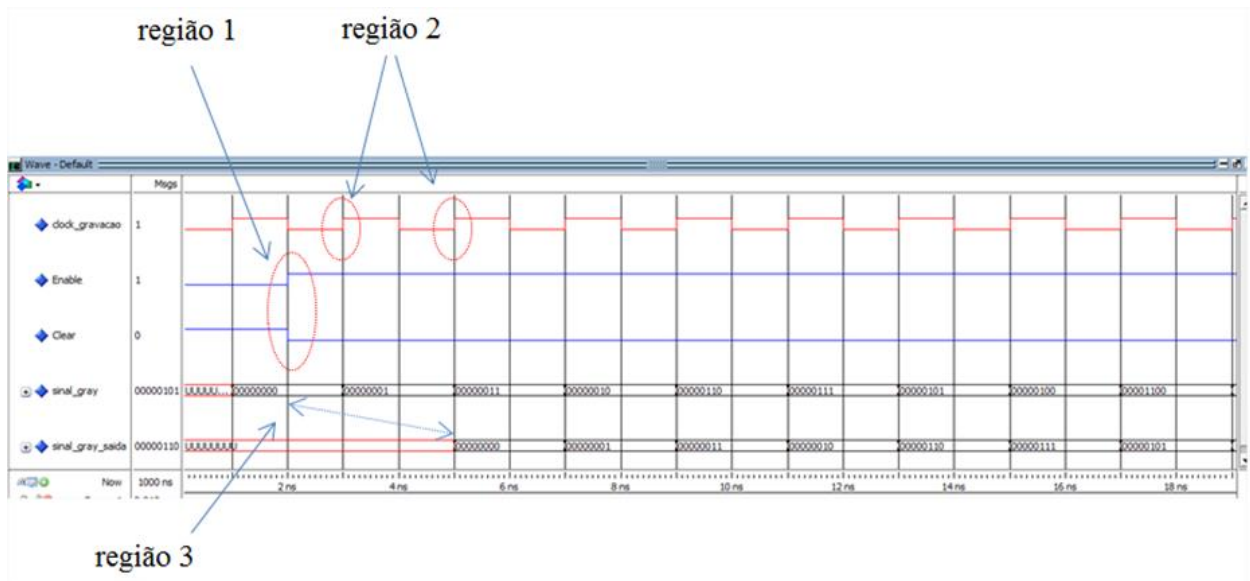


Figura 17 – Resultado do teste no módulo sincronismo

Analisando as regiões da Figura 17:

- Na região 1 a situação da ocorrência de pelo menos um clock de gravação, para inicializar o contador. As condições necessárias são **Enable**=’0’ e **Clear**=’1’. Depois de inicializado ocorre o **Enable**=’1’ e **Clear**=’0’, e logo em seguida as ocorrências dos demais clock de gravação.
- Na região 2 as mudanças de clock de gravação. Nota-se que as mudanças de um número de código Gray para outro número ocorrem na borda de subida do clock de gravação.
- Na região 3 é observado o sincronismo, onde o primeiro clock é responsável de armazenar os sinais temporariamente e na ocorrência do segundo clock o resultado é enviado para a saída **GrayCount_out**.

2.2.7. Cenário 4: Metaestabilidade

A metaestabilidade é outro cenário de dificuldade encontrado neste trabalho, e esperasse ser resolvido em trabalhos futuros. A metaestabilidade ocorre quando há concorrência, frequência de clock de leitura e gravação muito próximo, criando um ambiente crítico. A metaestabilidade deve ser evitada a todo custo. A simulação deste erro não pode ser desenvolvida no simulador ModelSim, pois existem diversos fatores que interferem no resultado, por exemplo a temperatura do ambiente, atraso dos fios, capacitância, concorrência

de clock muito próximo, entre outros. O resultado deste erro é indesejável [3], pois interfere diretamente na qualidade do dado. Logo, o valor armazenado após esta interferência pode estar corrompido, demorando em aparecer por causa da falha de temporização de clock, pois valores errados surgem no armazenamento, sendo indefinido e oscilando entre 0 e 1.

2.5. Dificuldades e Limitações

Durante o estágio desenvolvido pelo aluno foram encontradas diversas dificuldades. Neste trabalho foi usada a “versão estudante do ALTERA” que é de uso limitado. Assim, a implementação da arquitetura FIFO foi feita com as funções e métodos básicos disponíveis nesta versão. Vale ressaltar que a empresa ALTERA possui sua própria versão de DC-FIFO embutida, porém é de uso comercial e código fechado.

Por outro lado, neste projeto não foram usadas as mega funções do próprio Quartus II que pertencem ao kit MegaWizar de plug-in Manager, pois nesses módulos só é possível ter acesso nos pinos de entrada e saída, desconhecendo-se a sua implementação. Neste projeto foram desenvolvidos e testados todos os componentes separadamente, e logo depois foram integrados os módulos à montagem do projeto.

CAPÍTULO 3: CONCLUSÃO

O presente trabalho desenvolvimento de uma estrutura FIFO dual clock em FPGA como parte do estágio na empresa FIT. Além disso, foi apresentada uma bateria de testes que validaram a estrutura no ambiente de simulação. A seguir serão dadas as contribuições e trabalhos futuros.

3.1. Contribuições

Destaca-se que a implementação desenvolvida neste estágio foi feito em nível de hardware, apenas com as primitivas e funções básicas do QUARTUS II, ganhando em rapidez e tempo de execução. Além disso, foram desenvolvidos módulos de implementação FIFO com uma estrutura compreensível, permitindo ter o controle desta e facilidade para posteriores modificações, que não poderia ser feito se acaso fossem usadas funções prontas do ALTERA. Por outro lado, o estagiário tem contribuindo com o propósito geral da empresa, pois posteriormente, a FIFO será integrada ao projeto maior objetivado pela CIERMag e a FIT.

Finalmente o trabalho realizado contribuiu para o desenvolvimento profissional e pessoal do aluno. Profissionalmente o aluno adquiriu conhecimento em programação VHDL, FPGA, entre outras, o qual ainda permitirá ao aluno continuar na empresa de estágio no decorrer do ano de contrato.

3.2. Relacionamento entre o Curso e o Projeto de Estágio

Os conhecimentos adquiridos durante o curso Bacharelado em Informática foram muito importantes para o desenvolvimento deste projeto. Especificamente foram envolvidas as disciplinas de Sistemas Digitais, Elemento de Lógica Digital, Arquiteturas de Computadores. Além disso, o estagiário esteve cursando a disciplina de Micro processador I como ouvinte no IFSC.

Os diversos conceitos envolvidos neste trabalho foram: a implementação dos circuitos e noções de circuitos combinacionais e sequências, entendimento de subsistemas lógicos, usam de memória flip-flop, registradores e contadores, projeto de circuitos lógicos sequenciais para a execução de instruções binárias, entre outros conhecimentos prévios.

3.3. Considerações sobre o Curso de Graduação

O curso de Bacharelado em Informática forneceu ao aluno muito conhecimento sobre a área de Tecnologia da Informação, a qual permitiu ao estagiário o desenvolvimento desse projeto. Os conhecimentos envolvidos relacionam as disciplinas de Sistemas Digitais, Elemento de lógica digital, arquiteturas de computadores, Microprocessador I, entre outras. Pelo que seria de grande importância que tivesse mais disciplinas relacionadas à área de hardware.

3.4. Trabalhos Futuros

Atualmente, a metaestabilidade representa um cenário muito complexo. Acredita-se que, em trabalhos futuros, uma vez que a análise e validação dos resultados da FIFO esteja completo e sem erros, poderia fazer-se uso real da estrutura na prática.

Como trabalho futuro, a FIFO poder ser estendida para intermediar a comunicação entre dois equipamentos qualquer, inclusive o espectrômetro, os quais operam em diferente frequência de clock de leitura e gravação.

REFERÊNCIAS

1. ALTERA, “*SCFIFO and DCFIFO Megafunctions*”. Disponível em:
<http://www.ALTERA.com/literature/ug/ug_fifo.pdf> último acesso em: 23 de outubro 2014.
2. FLOYD,T.L. “*Sistemas Digitais Fundamentos e aplicações*”, Capítulo 10: *Memória e Armazenamento*, pp. 552, Bookman 2007.
3. KASTENSMIDT, F. G. L. “*Introdução a Sistemas digitais*”. Metaestabilidade. Universidade Federal do Rio Grande do Sul. Disponível em:
<<http://www.inf.ufrgs.br/~fglima/aula19.pdf>> último acesso em: 23 de outubro 2014.
4. ORDONEZ, E. D. M.; PEREIRA, F. D.; PENTEADO, C. G.; PERICINI, R. A. “*Projeto, Desempenho e Aplicações de Sistemas Digitais em Circuitos Programáveis (FPGAs)*”, Capítulo 1: Introdução, pp. 10, Pompéia: Bless,2003.
5. ROSÁRIO, D. A. N. D. “*Desenvolvimento e aplicações de sistemas embarcados em FPGA*”. TCC da Universidade Federal do Rio Grande do Norte, 2010.
6. Terasic, “*SoCKit User Manual, (rev. B Hardware)*”, Disponível em:
<<http://tinyurl.com/narra9g>> último acesso em : 23 de outubro 2014.
7. TOCCI, RONALDS J. “*Sistemas digitais: princípios e aplicações/ Ronald J. Tocci, Neal S. Widmer, Gregory L. Moss*”, Capítulo 2- Sistemas de Numeração e códigos. 11. ed., São Paulo: Pearson Prentice Hall, 2011.

APÊNDICE A – O POTENCIAL DA FPGA

A FPGA são matrizes de portas programáveis, as quais permitem a implementação de circuitos digitais. A vantagem de tratar com hardware programável, é que possibilita realizar números gigantes de instruções por cada ciclo de clock. Por isto, podem ser implementadas várias funções lógicas ou até mesmo montar um processador na FPGA, é dizer, podem-se instanciar vários processadores e rodar em paralelo, entregando resultados dos cálculos a cada ciclo de clock.

A FPGA é muito usada em diferentes setores elétricos e telecom, em switches e roteadores, para o processamento digital (imagens, etc.) que exige resultado em tempo real de e alto desempenho. A FPGA atinge um máximo de desempenho e confiabilidade nos resultados das funções lógica programáveis, devido a que não possui um sistema operacional rodando junto.

Na tabela 3 foi feita uma comparação entre as vantagens de usar a FPGA ao invés de microprocessadores.

Tabela 3 – Microprocessador versus FPGA

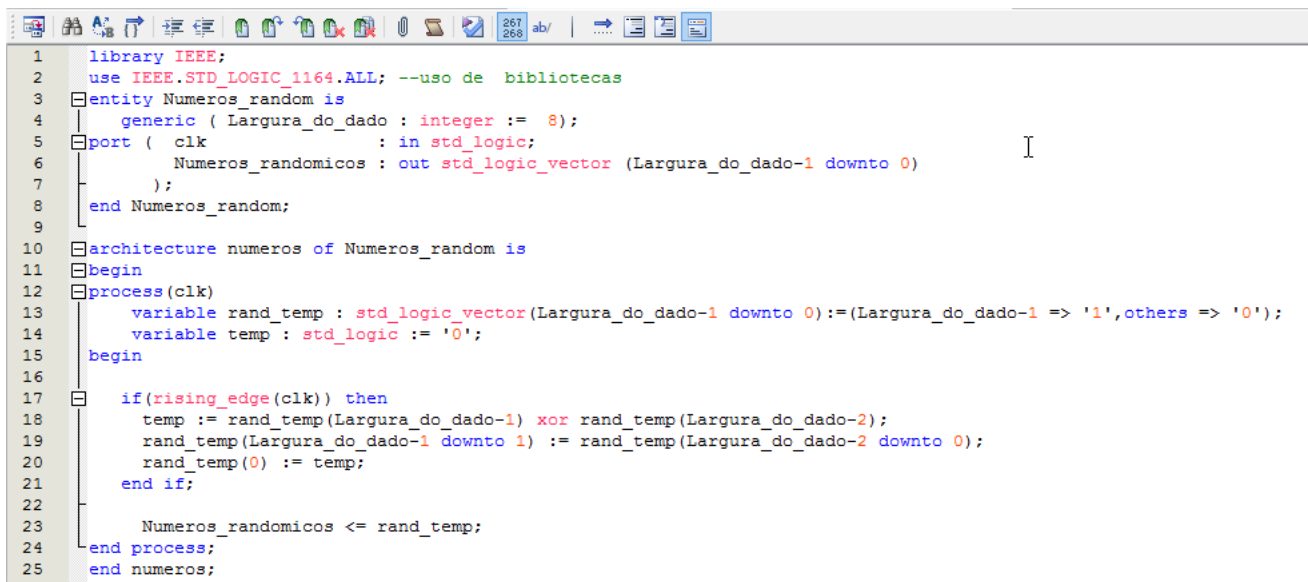
Microprocessadores	FPGA
Execução serial: A execução é de linha após linha, seguindo uma lógica sequencial.	Execução paralela: tudo acontece ao mesmo tempo, a cada ciclo de clock.
Desempenho cai quando a aplicação cresce porque o programa é sequencial e tem a dependência de instruções.	Sem lentidão conforme a aplicação cresce. A cada ciclo de clock executa inúmeras intrusões em paralelo.
O sistema operacional executa a lógica de controle (prioridade de recurso).	A lógica de controle em hardware é dedicada.
Módulos de E/S possui funcionalidade fixa.	Funcionalidade de E/S e reprogramável

APÊNDICE B – MÓDULO DE NÚMEROS RANDÔMICOS

Para fins de compreensão mostraremos a montagem completa do módulo de números randômicos para fins de exemplificação. Note-se que os demais módulos desenvolvidos neste projeto seguiram esta mesma rotina de implementação. É dizer, a validação, visualização do RTL, a implementação da bateria de teste, a integração, e finalmente o mapeamento e ligação das entradas e saídas virtuais para as pinagem reais da FPGA, como mostrado no ciclo de desenvolvimento mencionado na Seção 2.3.1.

B.1 Implementação

A seguir é mostrado o código em linguagem VHDL do módulo assim como sua respectiva arquitetura (ver Figura 18), logo depois serão descritas as demais etapas do ciclo de desenvolvimento.



```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL; --uso de bibliotecas
3  entity Numeros_random is
4      generic ( Largura_do_dado : integer := 8);
5      port ( clk : in std_logic;
6            Numeros_randomicos : out std_logic_vector (Largura_do_dado-1 downto 0)
7            );
8  end Numeros_random;
9
10 architecture numeros of Numeros_random is
11 begin
12 process(clk)
13     variable rand_temp : std_logic_vector(Largura_do_dado-1 downto 0):=(Largura_do_dado-1 => '1',others => '0');
14     variable temp : std_logic := '0';
15     begin
16
17     if(rising_edge(clk)) then
18         temp := rand_temp(Largura_do_dado-1) xor rand_temp(Largura_do_dado-2);
19         rand_temp(Largura_do_dado-1 downto 1) := rand_temp(Largura_do_dado-2 downto 0);
20         rand_temp(0) := temp;
21     end if;
22
23     Numeros_randomicos <= rand_temp;
24 end process;
25 end numeros;
```

Figura 18 – Implementação do módulo de números randômicos em VHDL e respectiva arquitetura.

- Linha 1: Declaração das bibliotecas padrão **library IEEE**.
- Linha 2: Uso do pacote **use ieee.std_logic_1164.all** para podemos trabalhar com o bit e o vetor de bit

- Linha 4: Variável 'global' para este módulo (**generic**), nela é possível modificar o tamanho da estrutura do vetor com passagem de parâmetro. Este valor é configurável na entidade *main*, no módulo principal (controle.vhdl)
- Linha 5-7: Na **port** é declarado todas as entradas e saídas, define-se a porta e o nome em seguida a **architecture** é implementado a lógica comportamental desta unidade e fazendo usos de combinação de portas lógicas xor.
- Linha 12-24: Fazendo uso de **process**, onde o clock é usado como lista de sensibilidade a cada modificação, ou seja, nas mudanças de 0 para 1 ou vice-versa, este processo é disparado. Logo em seguida são declarados os tipos **variable**, os quais são restritos a esse processo, ou seja, só pode ser modificada dentro desse processo. A variável **rand_temp** e **temp** são inicializados como zero logo no primeiro clock e a função lógica só será validada na borda de subida do clock, quando muda de 0 pra 1.

Os cálculos para gerar números randômicos, com o uso da porta lógica xor a cada bit do vetor, com um deslocamento do tamanho -1 e com o mesmo de tamanho -2 como segue na Figura 19, neste exemplo o tamanho é igual a 8.

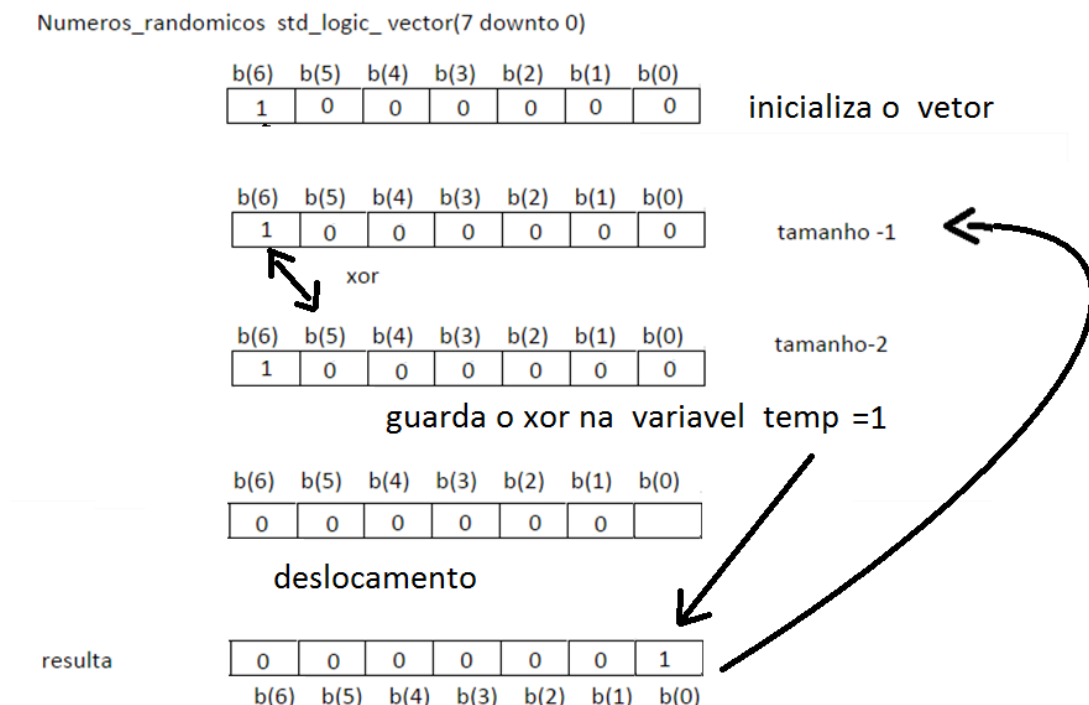


Figura 19 – Algoritmo de cálculo de números randômicos para 8 bits

Logo em seguida ocorre uma **latch** (flip-flop), captando os dados na entrada D e reenviando para o novo xor na saída Q, que é usada para a saída definitiva do módulo e serve

como entrada de um novo xor do mesmo vetor. As mudanças ocorrem a cada borda de subida do clock.

Linha 17: No final da condicional **IF** ocorre atribuição do resultado na variável de saída **números_randomicos**, que está no final do processo.

Este módulo de números randômicos representa a um emissor (espectrômetro) que gera sinais pseudoaleatórios enviando-os para ser armazenados na FIFO.

B.2 Visão estrutural em RTL

Na Figura 20 mostra-se a interpretação lógica programável na visão RTL do Quartus II, que é uma visão do ambiente de simulação do projeto.

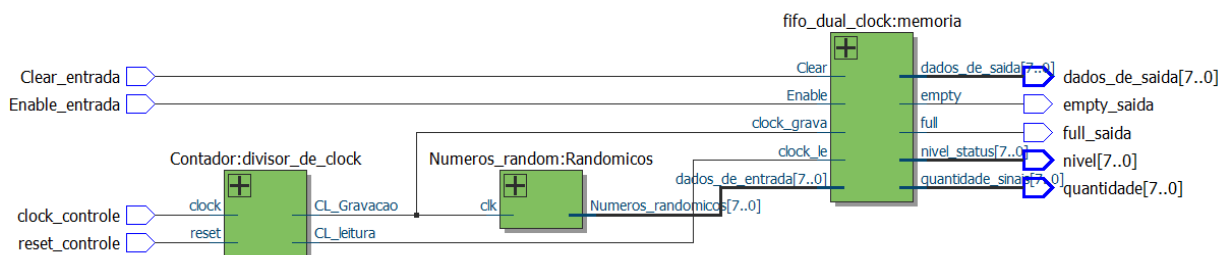


Figura 20 – Visão em RTL do Projeto

Na figura 21, mostra uma visão que inclui todos os módulos, indica a área de uso do dispositivo e como está mapeado no *chip planner*. Observa-se as regiões (marcadas em círculos) onde estão mapeados os ambientes de teste da FIFO, incluindo ela mesma. Também é observado que algumas funções lógicas estão ocupando regiões diferentes.

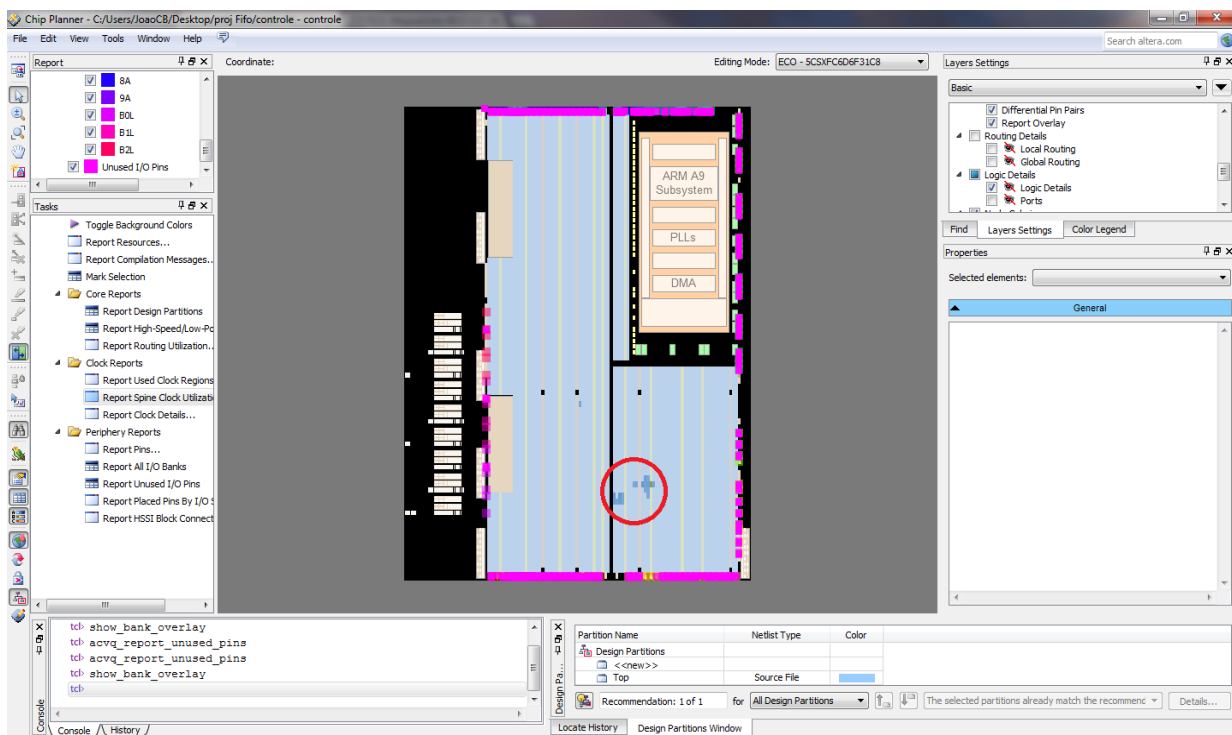


Figura 21 – Área em uso do projeto do *chip planner*

A Figura 22 mostra uma visão detalhada de como está implementado a área de uso à nível de portas lógicas. Nota-se que os módulos estão mapeadas em regiões diferente da FPGA.

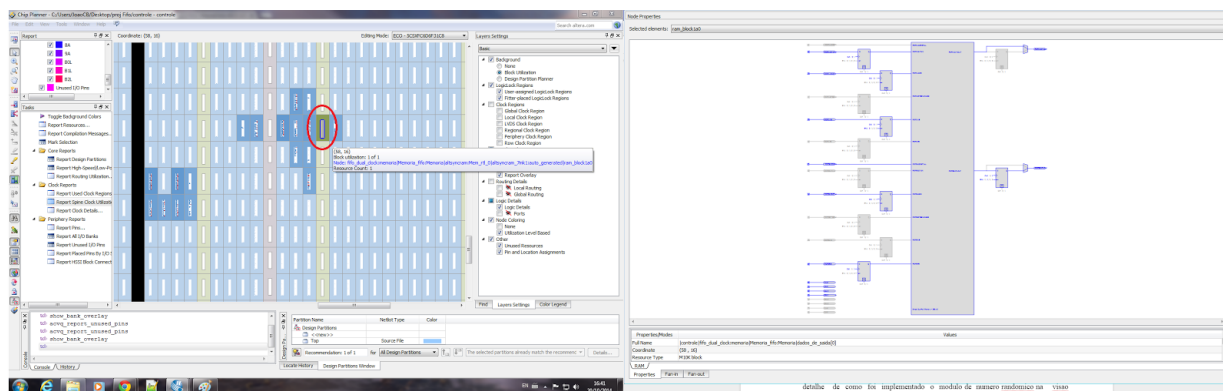


Figura 22 - Chip Planner : área em uso em detalhes

A Figura 23 mostra a visualização esquemática da estrutura interna do módulo de números randômicos. Nesta visão relata detalhes de portas lógicas usadas neste módulo.

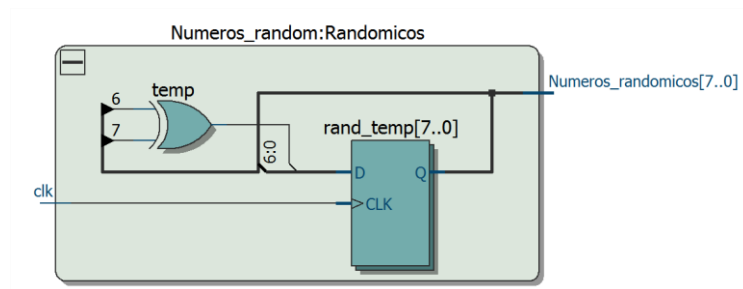


Figura 23 – Visão interna do módulo Números randômicos em RTL

B.3 Implementação da bateria de teste (*test bench*)

Aqui é feita a implementação da estrutura de teste. A montagem do *test bench* segue uma sequencia de linhas de código conforme é mostrado na Figura 24.

```

1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3
4  ENTITY test_bench_numeros_randomicos IS -- sem pinos I/O
5  |   END test_bench_numeros_randomicos; -- entidade sem corpo
6  |
7  ARCHITECTURE randomicos OF test_bench_numeros_randomicos IS
8  |
9  |   component Numeros_random               --instanciando o modulo
10 | port (
11 |     clk                : in std_logic;
12 |     Numeros_randomicos : out std_logic_vector (7 downto 0)
13 | );
14 |   END component;
15 |
16 |   signal clock_entrada: std_LOGIC:='0'; -- sinais de entrada
17 |   signal Numeros      : std_logic_vector (7 downto 0); -- resultado da saida
18 | BEGIN -- As entradas e saídas são observadas pelo simulador
19 |
20 |   clock_entrada<=not(clock_entrada) after 10 ns; -- conjunto de estímulos
21 |
22 |   teste_saida: Numeros_random -- testando o modulo
23 | PORT map( clock_entrada,
24 |           Numeros
25 | );
26 | END randomicos;

```

Figura 24 – Implementação do *test bench*

Vale ressaltar que a entidade do módulo *test bench* é declarada sem os pinos de entradas e saídas.

1º : Declarar a entidade sem corpo. Depois de criada a entidade, deve-se declarar um nome para a arquitetura que montará o teste.

2º : Instanciar o módulo a ser testado (**números_randomicos**)

3º: Declarar os sinais de estímulos de entradas e saídas para testar o módulo.

4º: Montar uma combinação de sinais de entrada para testar o módulo.

5º : Testar o módulo com os sinais de entradas e analisar os sinais de saída.

6º : Acompanhar os resultados usando o Simulador ModelSim.

B.3 Simulação e Validação

A Figura 25 é o resultado da simulação usando o simulador ModelSim. Observa-se que a cada ciclo de clock, foram gerados novos números randômicos.

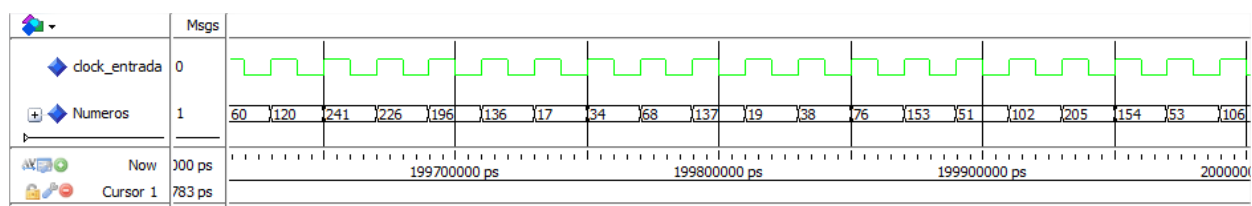


Figura 25 – Resultado do test banch em Números randômicos

Este gerador de números aleatórios pode representar o uso de um equipamento (espectrômetro) enviando sinais para ser armazenada na FIFO, a certa frequência de gravação o que está representado pelo módulo divisor de clock. Este simula a frequência de trabalho do emissor e também do receptor. Logo após verificarmos que o resultado do módulo desenvolvido atingiu um comportamento desejável, seguindo a lógica prevista. Logo depois, podemos proceder à integra-lo no projeto todo.

B.4 Integração dos módulos testados

Depois de verificarmos os testes de cada componente separadamente, podemos integra-lo e ver o resultado de todo o projeto no simulador ModelSim. No próximo passo é mostrado a configuração das pinagens de entrada e saída reais na FPGA (ver Figura 26).

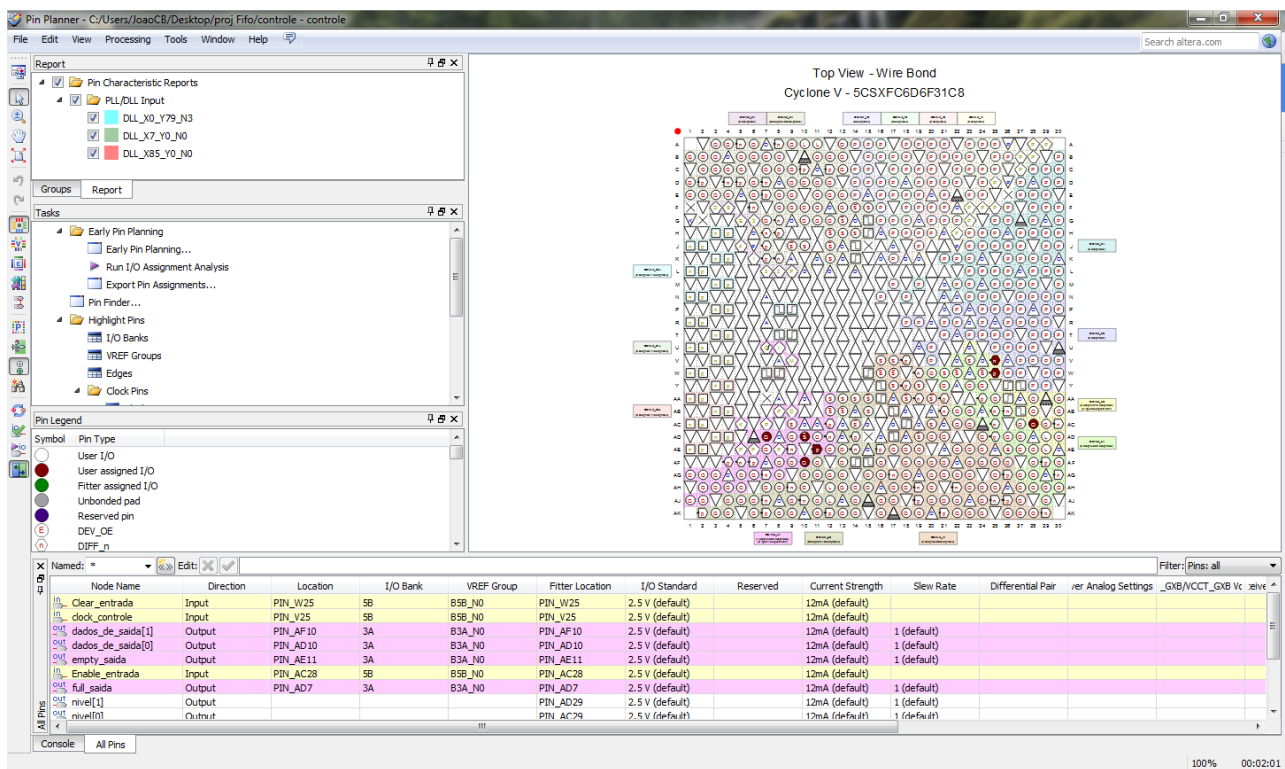


Figura 26 – Pin Planner

No manual da placa [5], encontra-se a tabela que mostra o mapeamento dos pinos de entrada e saída da FPGA, onde é mostrado o nome do pino (Switch, Push-button, led e outros), código da pinagem e as descrições respectivas. Deve-se ligar :

- Associar as Entradas reais da FPGA com as entradas virtuais do projeto.
- Associar as saídas reais da FPGA com as saídas virtuais do projeto.

Depois de ser compilado todo o projeto no Quartus, é dizer, é gerado o “arquivo.sof”, no qual encontra-se toda a síntese do projeto em portas lógicas.

B.5 Pinagem dos pinos virtuais para os pinos reais da FPGA

O projeto (“arquivo.sof”) é gravado na memória RAM da FPGA de forma volátil. A Figura 27 mostra este processo de gravação. Finalmente, podem ser feitos os testes na placa FPGA. É importante observar que caso haja a falta de energia na FPGA pode ocorrer a perda do projeto.

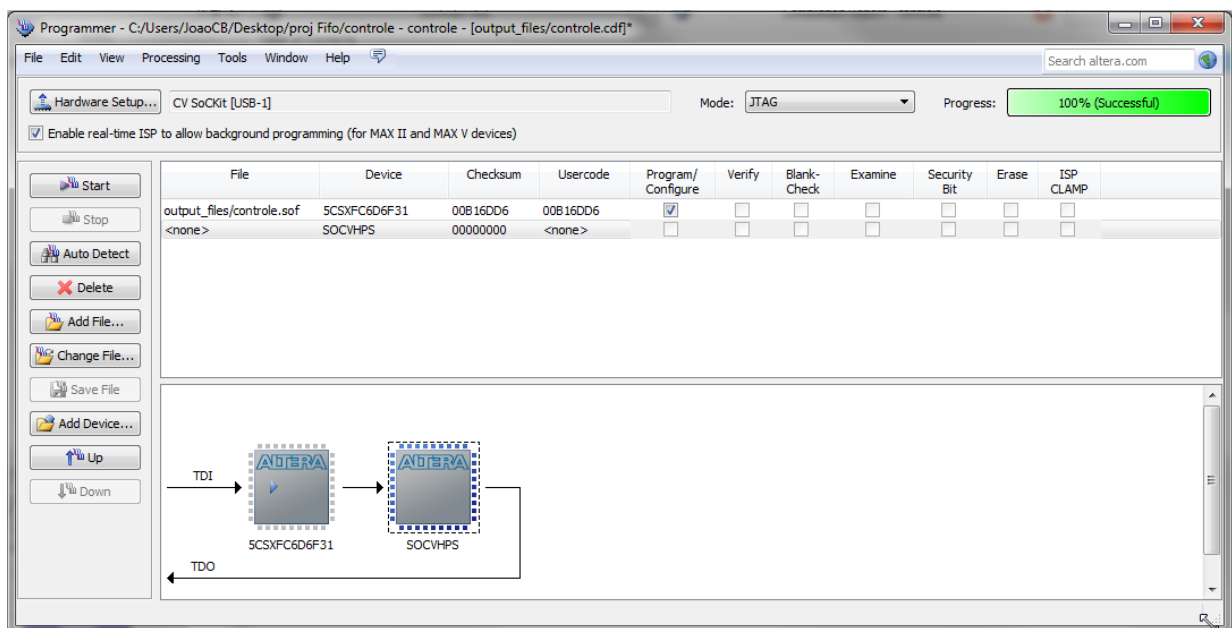


Figura 27 – Gravando o projeto na FPGA

APÊNDICE C – Implementação do módulo controlador de quantidade

Neste Apêndice é mostrado o código em VHDL do módulo Controlador de quantidade, como foi mencionado na Seção 2.3.4.

```

1      library ieee;
2      use ieee.std_logic_1164.all;
3      use ieee.std_logic_unsigned.all;
4      use ieee.std_logic_arith.all;
5
6      entity quantidade is
7      generic ( Largura_do_dados      : positive; -- é reconfigurado na main
8                profundidade         : positive -- controle
9      );
10     port (
11         clock_gravacao      :in std_logic;
12         clock_leitura       :in std_logic;
13         Enable_in           :in std_logic;      -- '0011000 ... 011'
14         quantidade_fifo     :out std_logic_vector (Largura_do_dados-1 downto 0);
15         nivel               :out std_logic_vector (Largura_do_dados-1 downto 0);
16         vazio              :out std_logic;
17         cheio              :out std_logic;
18         Enable_leitura      :out std_logic;
19         Enable_Gravacao     :out std_logic;
20     );
21 end entity;
22
23
24 architecture rst of quantidade is
25     -- uso de sinais dentro do modulo
26     constant profundidade_fifo : positive := 2**profundidade; -- capacidade da FIFO
27
28     signal somar                : std_logic_vector(Largura_do_dados-1 downto 0);
29     signal incrementar_gravacao : std_logic_vector(Largura_do_dados-1 downto 0):=(others=>'0');
30     signal incrementar_leitura  : std_logic_vector(Largura_do_dados-1 downto 0):=(others=>'0');
31
32     signal grava,Enable_G,Enable_L,le : std_LOGIC;
33
34 begin
35
36     indicador_de_nivel: process (clock_gravacao,clock_leitura)
37     begin -- somar resulta : somar <= incrementar_gravacao - incrementar_leitura;
38         if (unsigned(somar) < (profundidade_fifo/4)) then
39             nivel<=conv_std_logic_vector(1,Largura_do_dados);
40         end if;
41
42         if (unsigned(somar) > (profundidade_fifo/4) and unsigned(somar) < (profundidade_fifo/2)) then
43             nivel<=conv_std_logic_vector(2,Largura_do_dados);
44         end if;
45
46         if (unsigned(somar) > (profundidade_fifo/2) and unsigned(somar)<(3*(profundidade_fifo/4))) then
47             nivel<=conv_std_logic_vector(3,Largura_do_dados);
48         end if;
49
50         if (unsigned(somar) > (3*(profundidade_fifo/4)) ) then
51             nivel<=conv_std_logic_vector(4,Largura_do_dados);
52         end if;
53     end process;
54
55     Estado_da_fifo: process (somar) -- indica fifo cheio ou vazio
56     begin
57         vazio<='0';
58         cheio<='0';
59
60         if ( unsigned(somar) < 1 ) then -- empty
61             vazio<='1';
62         else
63             vazio<='0';
64         end if;
65
66         if ( unsigned(somar) >= (profundidade_fifo-1) ) then -- full
67             cheio<='1';
68         else
69             cheio<='0';
70         end if;
71     end process;
72
73 end process;
74

```

```

74 |
75 | incrementa_gravacao: process (clock_gravacao)
76 | begin
77 |     if (clock_gravacao'EVENT AND clock_gravacao = '1') THEN
78 |
79 |         if( Enable_G = '1' and grava = '1' )THEN
80 |             incrementar_gravacao<=incrementar_gravacao+1;
81 |         END IF ;
82 |
83 |     END IF ;
84 |
85 | end process;
86 |
87 | incrementa_leitura: process ( clock_leitura)
88 | begin
89 |     if ( clock_leitura'EVENT AND clock_leitura = '1' ) THEN
90 |
91 |         if( Enable_L = '1' and le = '1' )THEN
92 |             incrementar_leitura <= incrementar_leitura+1;
93 |         END IF ;
94 |
95 |     END IF ;
96 |
97 | end process;
98 |
99 |     somar <= incrementar_gravacao - incrementar_leitura;
100 |
101 | controla_processo_de_leitura_gravacao: process (clock_gravacao,clock_leitura)
102 | begin
103 |     grava<='1';
104 |     le<='1';
105 |
106 |     if( unsigned(somar) >= (profundidade_fifo-1) )THEN -- fifo cheio nao grava
107 |         grava<='0';
108 |     else
109 |         grava<='1';
110 |     END IF;
111 |
112 |     if( unsigned(somar) > 0 )THEN -- fifo vazio nao le
113 |         le<='1';
114 |     else
115 |         le<='0';
116 |     END IF;
117 |
118 | end process;
119 |
120 | semafaro_de_leitura_gravacao:process (clock_gravacao,clock_leitura)
121 | begin
122 |     Enable_G<='1'; -- sinais para controle dentro deste modulo
123 |     Enable_L<='1';
124 |     Enable_Gravacao<='1'; -- sinais de controle enviado para fora do modulo
125 |     Enable_Leitura<='1';
126 |     if ( unsigned(somar) >= (profundidade_fifo-1) and Enable_in='1' ) THEN-- desabilita gravacao
127 |         Enable_G<='0'; -- fifo cheia
128 |         Enable_L<='1'; -- habilita leitura
129 |         Enable_Gravacao<='0';
130 |         Enable_Leitura<='1';
131 |     elsif( unsigned(somar) < 1 and Enable_in='1' )then -- desabilita leitura
132 |         Enable_G<='1'; -- fifo vazia
133 |         Enable_L<='0'; -- habilita gravacao
134 |         Enable_Gravacao<='1';
135 |         Enable_Leitura<='0';
136 |     END IF ;
137 |
138 |     if( Enable_in = '0' )then -- desabilita leitura e gravacao
139 |         Enable_G<='0';
140 |         Enable_L<='0';
141 |         Enable_Gravacao<='0';
142 |         Enable_Leitura<='0';
143 |     END IF ;
144 |
145 | end process;
146 |
147 |     quantidade_fifo <= somar; -- indica o quanto
148 |
149 | end rst;
150 |

```